

**REMARKS**

In light of the above amendments and following remarks, reconsideration and allowance of this application are respectfully requested.

At paragraph 3 of the outstanding Office Action, the Examiner has objected to the specification citing various informalities. Applicants have addressed these informalities as suggested by the Examiner, and therefore request that the objection to the specification on these grounds be withdrawn.

At paragraph 4 of the outstanding Office Action, the Examiner has requested a new title. While Applicants have not used the title suggested by the Examiner, Applicants have amended the title to one that is more clearly indicative of the invention to which the claims are directed.

At paragraph 5 of the outstanding Office Action, the Examiner has requested amendments to the Abstract. Applicants have amended the Abstract to more particularly set forth the goal of the invention. Applicants therefore request that the objection to the Abstract be withdrawn.

The Examiner has requested changes to Figs. 1, 4, 6 and 7. Applicants have amended these figures as requested by the Examiner, and therefore request that the objection to the drawings on these grounds be withdrawn. Applicants note that the addition of databases to the portal of figure 1 would be known in the art for storing various scripts, graphics and other objects, as noted at page 2, lines 1-3 of the application.

At paragraph 7 of the outstanding Office Action, the Examiner has indicated that no International Search Report was provided with the European references. Applicants note that

the four references submitted in the Information Disclosure Statement are in the English language, and therefore a copy of the European Search Report is not required. Applicants submit concurrently herewith two prior art references (“Model-View-Controller,” May 14, 1998, from <http://ootips.org>; and Model View Controller, from <http://www.object-arts.org>) describing the well-known model-view controller.

At paragraph 8 of the outstanding Office Action, the Examiner has rejected claims 1-8 under 35 U.S.C. §101 as presenting software that is not tangibly embodied on a computer readable medium. Applicants have amended independent claims 1 and 6 to specifically recite that the portal application is for implementation on a multipurpose computer, and therefore request that the rejection to these claims be withdrawn.

At paragraph 9 of the outstanding Office Action, the Examiner has objected to the terms “core service” and “special service”, indicating that there is insufficient antecedent basis for these limitations in claim 9. Applicants note that in each case, sufficient antecedent basis was previously provided. Nonetheless, Applicants have amended independent claim 9 to recite “a first core service” and “a second special service”, thereby providing proper antecedent basis. Applicants therefore request that the rejection of claim 9 under 35 U.S.C. §112 be withdrawn.

At paragraphs 10-11, the Examiner has rejected claims 1-10 as using terms rendering the claims indefinite. Applicants have amended each of the claims to more particularly define the claims, specifically addressing to the portions of the claims noted by the Examiner. In light of these amendments to the claims, Applicants respectfully request that the rejection of claims 1-10 under 35 U.S.C. §112 be withdrawn.

At paragraphs 13-15, the Examiner has rejected claims 1-10 under 35 U.S.C. §102(e) as being anticipated by Helgeson et al. (U.S. Patent No. 6,643,652). Applicants respectfully traverse the rejection.

Each of independent claims 1, 6, 8 and 9 recite that the portal application is structured in accordance with a “model-view-controller architecture”. Applicants submit that this “model-view-controller” is a well known software implementation, and comprises a logical structure of a software program. Such a “model-view-controller” does not comprise a hardware arrangement. A more complete description of such a model-view controller is set forth in the two accompanying prior art references “Model-View-Controller” as described in the website “object-arts.com,” and “Model-View-Controller” as described at the website “ootips.org,” a copy of each of which have been included with the accompanying Information Disclosure Statement.

Applicants submit that Helgeson merely describes a three-tier hardware structure and in fact describes nothing more than a distributed network system. Applicants submit that Helgeson has nothing to do with a logical organization of a multi-media service according to a model-view-controller software model. Applicants submit that the present invention presents a unique application for using a model-view-controller approach as applicable to a portal application for providing accesses from clients to a multi-media service.

Applicants therefore respectfully request that the rejection of claims 1-10 under 35 U.S.C. §102(e) be withdrawn.

In the Conclusion at paragraph 16, the Examiner further notes that the Examiner has found numerous arts that overcome the disclosed subject matter, and also makes a note that all the claims 1-10 are anticipated by the references cited by the International Search Report. It is unclear whether the Examiner intended to reject the pending claims over these references. Applicants request that a comparison between the claimed elements and the references be provided in proper rejection.

Nonetheless, Applicants respectfully disagree with the Examiner’s interpretation of the references cited in the International Search Report as applied to the present claims. Applicants

note that in each situation, and particularly the Betz article, while a model-view-controller model is depicted, the teaching is limited to the interaction between a single client and a server. However, in accordance with the invention, for example as set forth in claim 1, a plurality of services are provided, and therefore allow for connection from a client via the model-view-controller model for accessing various services. Thus, while Betz suggests a one-to-one correspondence, the claimed invention refers to a many-to-one correspondence. Thus, according to the present invention, interaction between a client and different services is permissible. As the services communicate with each other by means of a controller, services can be dynamically installed and removed as necessary.

Applicants therefore submit that while the Examiner has not formally rejected the claims based upon the references in the International Search Report, doing so would be improper.

**CONCLUSION**

Applicants have made a diligent effort to place claims 1-10 in condition for allowance, and notice to this effect is earnestly solicited. If the Examiner is unable to issue a Notice of Allowance regarding claims 1-10, the Examiner is respectfully requested to contact the undersigned attorney in order to discuss any further outstanding issues.


Early and favorable consideration are respectfully requested.

Please charge any fees incurred by reason of this response to Deposit Account No. 50-0320.

Respectfully submitted,

FROMMER LAWRENCE & HAUG LLP  
Attorneys for Applicant(s)

By:

  
Gordon M. Kessler  
Reg. No. 38,511  
(212) 588-0800



# Model-View-Controller

Source: [comp.object](#)  
Date: 14-May-98

Search:

[Related Sites](#)

## OO Books



### New:

- ◆ [Data Model for OO Applications](#)
- ◆ [Refactoring](#)
- ◆ [Pitfalls of Process Frameworks](#)
- ◆ [Translation of Object Models to Code](#)
- ◆ [Estimating OO Projects](#)

### Popular:


- ◆ [Extreme Programming](#)
- ◆ [Model-View-Controller](#)
- ◆ [Hungarian Notation](#)
- ◆ [When Are Use Cases Done?](#)
- ◆ [ER Diagrams and OO](#)

[All the tips...](#)



[Send feedback](#)

 **Problem:** The Model-View-Controller (MVC) is a commonly used and powerful architecture for GUIs. How does it work?

 **Dean Helman** wrote (an extract from Objective Toolkit Pro whitepaper):

The MVC paradigm is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller. MVC was originally developed to meet the traditional input, processing, output roles into the GUI realm:

Input --> Processing --> Output  
Controller --> Model --> View

The user input, the modeling of the external world, and the visual feedback to the user are separated and handled by model, viewport and controller objects. The controller interprets mouse and keyboard inputs from the user and maps these user actions into commands that are sent to the model and/or viewport to effect the appropriate change. The model manages one or more data elements, responds to queries about its state, and responds to instructions to change state. The viewport manages a rectangular area of the display and is responsible for presenting data to the user through a combination of graphics and text.

[...]



RECEIVED

MAY 07 2004

Technology Center 2100

[The model is used] to manage information and notify observers when that information changes. [...] It contains only data and functionality that are related by a common purpose [...]. If you need to model two groups of unrelated data and functionality, you create two separate models.

[...] a model encapsulates more than just data and functions that operate on it. A model is meant to serve as a computational approximation or abstraction of some real world process or system. It captures not only the state of a process or system, but how the system works. This makes it very easy to use real-world modeling techniques in defining your models. For example, you could define a model that bridges your computational back-end with your GUI front-end. In this scenario, the model wraps and abstracts the functionality of a computation engine or hardware system and acts as a liaison requesting the real services of the system it models.

[...]

The [view or viewport] is responsible for mapping graphics onto a device. A viewport typically has a one to one correspondence with a display surface and knows how to render to it. A viewport attaches to a model and renders its contents to the display surface. In addition, when the model changes, the viewport automatically redraws the affected part of the image to reflect those changes. [...] there can be multiple viewports onto the same model and each of these viewports can render the contents of the model to a different display surface.

[...]

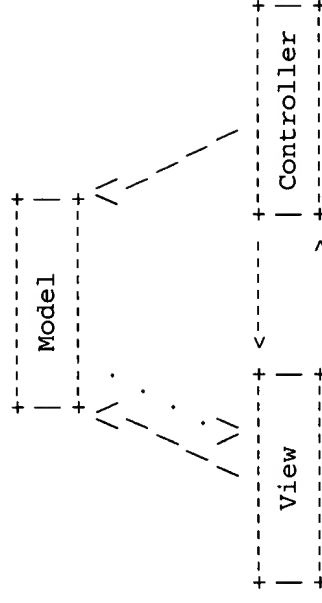
[A viewport] may be a composite viewport containing several sub-views, which may themselves contain several sub-views.

[...]

A controller is the means by which the user interacts with the application. A controller accepts input from the user and instructs the model and viewport to perform actions based on that input. In effect, the controller is responsible for mapping end-user action to application response. For example, if the user clicks the mouse button or chooses a menu item, the controller is responsible for determining how the application should respond.

[...]

The model, viewport and controller are intimately related and in constant contact. Therefore, they must reference each other. The picture below illustrates the basic Model-View-Controller relationship:



The figure above shows the basic lines of communication among the model, viewport and controller. In this figure, the model points to the viewport, which allows it to send the viewport weakly-typed notifications of change. Of course, the model's viewport pointer is only a base class pointer; the model should know nothing about the kind of viewports which observe it. By contrast, the viewport knows exactly what kind of model it observes. The viewport also has a strongly-typed pointer to the model, allowing it to call any of the model's functions. In addition, the viewport also has a pointer to the controller, but it should not call functions in the controller aside from those defined in the base class. The reason is you may want to swap out one controller for another, so you'll need to keep the dependencies minimal. The controller has pointers to both the model and the viewport and knows the type of both. Since the controller defines the behavior of the triad, it must know the type of both the model and the viewport in order to translate user input into application response.

 [More Info:](#)

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad,  
[Pattern Oriented Software Architecture: A System of Patterns](#) 

(contains detailed description of MVC, and also of a related architecture: Presentation-Abstraction-Control)



Ralph Johnson, [Model-View-Controller as an Aggregate Design Pattern](#)

Steve Burbeck, [How to use Model-View-Controller](#)

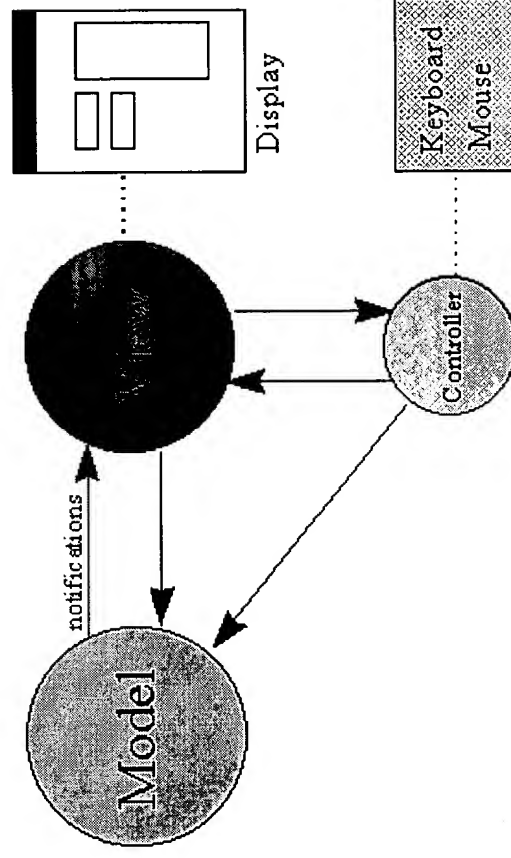
Object Arts, [Model View Controller](#)

Stingray Objective Toolkit Pro, an MVC implementation for MFC

# Model View Controller

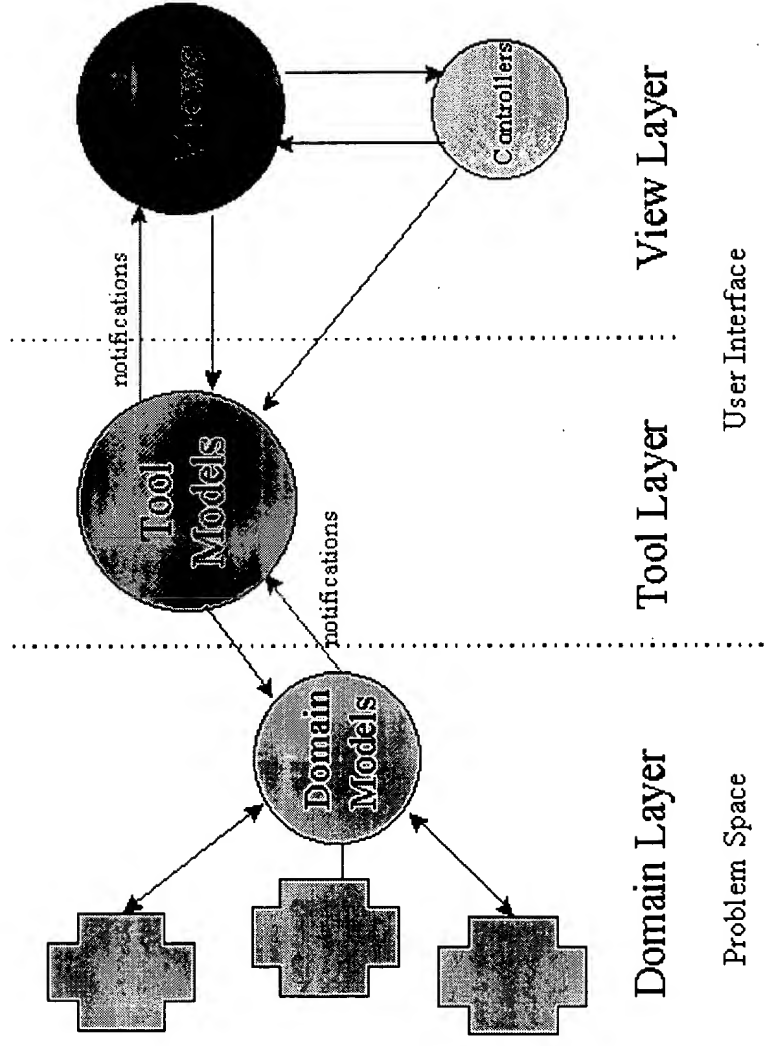
In Smalltalk-80™ the user interface was created around a framework known as Model-View-Controller or MVC. Over time this has been seen as a classic framework but some deficiencies have surfaced.

MVC applications are split into several triads each of which comprises a relationship between a Model object, a View object and a Controller. This interaction looks as follows.



The red link denotes dependency notifications. Traditional MVC implementations have further split the Model into Domain Models and Application Models which are also loose coupled using the dependency mechanism. The requirement was certainly present for something to sit between the View&Controller pair (which are usually quite generic) and the Domain Model which is business specific. The Application Model (or Tool) layer is therefore allowed to "know" something about the user interface while it manipulates the domain.

The resultant architecture looks as follows.



## Problems

Increasingly we came across problems with this approach. Most were caused by the Observer connection between Application Model and view. The difficulty here is that, as we have already said, the AM is allowed to know about the existence of the View but not vice versa. For example, the AM may be responsible for enabling and disabling view widgets or changing colours to show validation errors and such like. However, because the coupling between it and the View is loose it becomes difficult for this sort of interaction to take place. VisualWorks™ users will be familiar with the roundabout:

*builder componentAt: 'aViewName'*

method of accessing Views from within the Application Model.

One of the problems here is that the dependency mechanism between tool and view is being used for two, rather different purposes. First it is being used to implement the Observer pattern. This allows several Views to observe a single Model and display changes to it while the Model

knows nothing about these Views. However, we have already seen that an AM needs to be able to communicate with its Views so the reason for using Observer is weakened here. As soon as you have to use *componentAt*: you've broken Observer.

Secondly, the dependency mechanism allows an adaptive coupling between the Model and View. We want to be able to plug different generic Views onto more specific Models in many different ways, for maximum re-use. Thus the interface between the two needs to be flexible or simple or both. This ability is implemented in VisualWorks™ using ValueModels. This is the real reason for the loose coupling between AM and View.

What we really want, though, is a tight coupling between AM and View but a loose coupling between Domain Model and View. Rotate the triad slightly and we get the Model-View-Presenter framework.